TIME AND ACTION LOCK FREEDOM PROPERTIES FOR TIMED AUTOMATA

Howard Bowman

Computing Lab, Univ of Kent at Canterbury, Canterbury, Kent, CT2 7NF, UK

H.Bowman@ukc.ac.uk

Abstract Time and action locks can arise freely in timed automata specification. We review techniques for avoiding the occurrence of timelocks, based on the Timed Automata with Deadlines framework. Then we present a notion of parallel composition which preserves action lock freeness, in the sense that, if any component automaton is action lock free, then the composition will also be action lock free.

Keywords: Timed Automata, Timed Systems, Timelocks.

1. INTRODUCTION

Deadlocks are the characteristic error situation arising in concurrent and distributed systems. In very general terms, they are states in which the system is unable to progress further.

Classically the term deadlock has been seen as synonymous with what we will call *action locks.* These are situations in which, how ever long time is allowed to progress, the system will never be able to perform an action. Such action locks often result from unmatchable action offers, e.g. when a component wishes to perform a synchronisation action, but is unable to because no other process can offer a matching synchronisation.

In the context of timed systems, new locking situations arise. In particular, in this paper, we will be working in an environment with two main types of locking situation. As a result of this, we have had to be careful with our choice of terminology. Thus, in this paper the term deadlock is the most general. It embraces action locks and the form of locking behaviour that comes with timed systems - timelocks.

Timelocks are situations in which, informally speaking, time is prevented from passing beyond a certain point. They are highly degenerate occurrences [4] because they yield a *global* blockage of the systems evolution. This is quite different from an action lock, which cannot affect the evolution of an independent process.

In particular, unless significant care is taken, the possible presence of timelocks is a major problem for the formal specification and analysis of time critical networks and distributed systems. This is especially the case when using timed automata, which are at the heart of current timed model checking technology, such as UPPAAL [1] and Kronos [7].

This problem was highlighted in [6] where a number of timelock errors were discovered in a timed automata model of a lip-synchronisation protocol. Furthermore, it was shown in [4] that, when using timed automata, even the simple task of defining a timeout in a communication protocol is hampered by the possible presence of timelocks.

In fact, the issue of whether timelocks are desirable or undesirable features of timed models remains a hotly debated topic. The standard argument in favour of models containing timelocks is that they represent specification inconsistencies (like logical contradictions) and that by discovering and eliminating them, specifications can be corrected. However, we take the contrary position for a number of reasons.

Firstly, detecting timelocks is, in fact, a difficult and expensive analysis task. The classic method is to show that a property such as the Kronos formula, $\texttt{init} \Rightarrow \Box\Diamond_{=1}(\texttt{true})$, holds over the specification. This is an unbounded liveness property which is one of the most difficult classes of formulae to check. Recent work by Tripakis [9] offers potential improvements in such analysis. However, his algorithm remains unimplemented and furthermore, such improvements will always be thwarted by systems with fundamentally large state spaces.

We are also strongly of the opinion that inconsistencies and contradictions fit in the domain of logical description, but are difficult to reconcile with behavioural specification techniques, such as Timed Automata (TA). Contradictions arise when conflicting *properties* are asserted / conjoined. However, although the mistake is frequently made, parallel composition of processes is not a property composition operator, rather its meaning is operational - two (or more) physical components are *run* in parallel. Error situations in behavioural techniques should have a behavioural / operational intuition that is justifiable in terms of real world behaviour. This is the case for action locks and livelocks. However, there is no real world counter-part for time stopping!

It is also important to emphasize that the situation with action locks and timelocks is, in this respect, a little different. In our opinion timelocks are highly counter-intuitive and thus we believe that constructively preventing the occurence of timelocks is essential. However, since action locks are not in the same way counter-intuitive, prevention is not in the same sense essential. Nonetheless investigating techniques which ensure action lock freeness is useful since it highlights forms of parallel composition that can be employed when building systems that are “correct by construction”.

Although pleasingly simple, as previously implied, timed automata have the weakness that timelocks can freely arise and in a number of different ways. Perhaps most problematically they can arise through the interplay of urgency and synchronous interaction. We argue that urgency is given too strong an interpretation in timed automata, in the sense that an action can be forced (i.e. it becomes urgent) even if it is not possible (i.e. is not enabled).

The first part of this paper presents a re-interpretation of synchronisation that weakens the effect of urgency and thus limits the occurrence of timelocks. The approach uses ideas from the Timed Automata with Deadlines (TADs) framework of Bornot and Sifakis [2, 3]. However, we adapt the TADs definitions in the manner we introduced in [4].

The timed prioritised choice features offered by the TADs framework yield the possibility that the dynamic enabling of “competing” transitions can be defined statically. Hence notions of parallel composition that preserve different dynamic properties can be investigated. In this vein we also present a notion of parallel composition which preserves action lock freeness, in the sense that, if any of the component TADs is action lock free then the parallel composition will also be action lock free. Thus, in this paper we develop notions of parallel composition which are strongly compositional in the sense that if component processes are time and action lock free, then the parallel composition of these process will also be time and action lock free.

Structure of Paper. Section 2 introduces basic notation. Then we tackle the issue of timelocks in section 3 and action locks in section 4. Finally, section 5 presents concluding remarks.

2. BACKGROUND

Notation. $HA = \{x?, x! \mid x \in CA\}$ is a set of *half* (or *uncompleted*) actions. CA is a set of *completed* (or internal) actions. These give a simple CCS or UPPAAL [1] style point-to-point communication. Thus, two actions, $x?$ and $x!$ can synchronise and generate a completed action x. $A = HA \cup CA$ is the set of *all* actions. We use a complementation notation over elements of A: $\overline{x} = x$ if $x \in CA$, $\overline{x?} = x!$ and $\overline{x!} = x?$.

R^+ denotes the positive reals without zero and $R^{+0} = R^+ \cup \{0\}$. C is the set of all clock variables, which take values in R^{+0}. CC is a set of clock constraints. Also if $C \subseteq \mathrm{C}$ we write CC_C for the set of clock constraints generated from clocks in C.

$\mathrm{V}_C = C \rightarrow \mathrm{R}^{+0}$ is the space of clock valuations for clocks in C.

Timed Automata. An arbitrary TA has the form: (L, l_0, T, I, C), where, L is a finite set of locations; $l_0 \in L$ is a *start location;* and C is the set of clocks of the TA.

$T \subseteq L \times \mathrm{A} \times CC_C \times \mathrm{P}(C) \times L$ is a transition relation. A typical element of T would be, (l_1, e, g, r, l_2) where l_1, l_2 are locations; e labels the transition; g is a guard; and r is a reset set. $(l_1, e, g, r, l_2) \in T$ is typically written, $l_1 \xrightarrow{e,g,r} l_2$, stating that the automaton can evolve from location l_1 to l_2 if the (clock) guard g holds and in the process action e will be performed and all the clocks in r will be set to zero.

$I : L \rightarrow CC_C$ is a function which associates an invariant with every location. Intuitively, an automaton can only stay in a state while its invariant is satisfied.

It is important to understand the difference between the role of guards and of invariants. In this respect we can distinguish between *may* and *must* timing. Guards express may behaviour, i.e. they state that a transition is possible or in other words *may* be taken. However, guards cannot "force" transitions to be taken. In contrast, invariants define must behaviour. This must aspect corresponds to *urgency,* since an alternative expression is that when an invariant expires, outgoing transitions must be taken straightaway.

Semantics. TA are semantically interpreted over transition systems which are triples, $(S, s_0, \Rightarrow)$, where, $S \subseteq L \times \mathrm{V}_C$ is a set of states; $s_0 \in S$ is a start state; and $\Rightarrow \subseteq S \times Lab \times S$ is a transition relation, where $Lab = \mathrm{A} \cup \mathrm{R}^+$. Transitions can be of one of two types: *discrete transitions,* e.g. (s_1, e, s_2), where $e \in \mathrm{A}$ and *time transitions,* e.g. (s_1, d, s_2), where $d \in \mathrm{R}^+$ and denotes the passage of d time units. Transitions are written: $s_1 \overset{e}{\Longrightarrow} s_2$ respectively $s_1 \overset{d}{\Longrightarrow} s_2$.

For a clock valuation $v \in \mathrm{V}_C$ and a delay d, $v{+}d$ is the clock valuation such that $(v+d)(c) = v(c) + d$ for all $c \in C$. For a reset set r, we use $r(v)$ to denote the clock valuation v' such that $v'(c) = 0$ whenever $c \in r$ and $v'(c) = v(c)$ otherwise. v_0 is the clock valuation that assigns all clocks to the value zero.

The semantics of a TA $A = (L, l_0, T, I, C)$ is a transition system, $(S, s_0, \Rightarrow)$, where $S = \{s' \in L \times \mathrm{V}_C \mid \exists s \in S,\ y \in Lab\,.\, s \overset{y}{\Longrightarrow} s'\} \cup \{[l_0, v_0]\}$, $s_0 = [l_0, v_0]$ and $\Rightarrow$ is defined by the following inference reles:-

$$\frac{l \xrightarrow{e,g,r} l' \quad g(v)}{[l,v] \overset{e}{\Longrightarrow} [l', r(v)]} \qquad \frac{\forall d' \leq d\,.\, I(l)(v + d')}{[l,v] \overset{d}{\Longrightarrow} [l, v + d]}$$

The semantic map which generates transition systems from TA is written $[\![\,]\!]$. Also, notice that our construction ensures that only reachable states are in S.

Parallel Composition. We assume our system is described as a vector of TA, denoted, $|A = |\langle A[1], ..., A[n]\rangle$ where $A[i]$ is a timed automaton. In addition, we let u, u' etc, range over the set U of vectors of locations, which are written, $\langle u[1], ..., u[n]\rangle$. We use a substitution notation as follows: $\langle u[1], ..., u[j], ..., u[n]\rangle[u[j]'/u[j]] = \langle u[1], ..., u[j]', ..., u[n]\rangle$ and we write $[u[j]'/u[j]]$ as $[j'/j]$ and $u[i_1'/i_1]...[i_m'/i_m]$ as $u[i_1'/i_1, ..., i_m'/i_m]$.

If $\forall i(1 \leq i \leq n) \,.\, A[i] = (L_i, l_{i,0}, T_i, I_i, C_i)$ then the product automaton, which characterises the behaviour of $|\langle A[1], ..., A[n]\rangle$ is given by, (L, l_0, T, I, C) where $L = \{\, |u \mid u \in L_1 \times ... \times L_n \}$, $l_0 = |\langle l_{1,0}, ..., l_{1,n}\rangle$, T is as defined by the following two inference rules, $I(|\langle u[1], ..., u[n]\rangle) = I_1(u[1]) \wedge ... \wedge I_n(u[n])$ and $C = C_1 \cup ... \cup C_n$.

$$\frac{u[i] \xrightarrow{x?, g_i, r_i} u[i]' \quad u[j] \xrightarrow{x!, g_j, r_j} u[j]'}{|u \xrightarrow{x, g_i \wedge g_j, r_i \cup r_j} |u[i'/i, j'/j]} \qquad \frac{u[i] \xrightarrow{x, g, r} u[i]' \quad x \in CA}{|u \xrightarrow{x, g, r} |u[i'/i]}$$

where $1 \leq i \neq j \leq |u|$. Note, we write $x \leq k \neq r \leq y$ in place of $x \leq k \leq y \;\wedge\; x \leq r \leq y \wedge\; k \neq r$.

Timelocks. We can formulate the notion of a timelock in terms of a testing process. Consider, if we take our system which we denote `System` and compose it in parallel with the timed automaton, `Tester`, shown in figure 1, where, since it is completed, the `zzz` action is independent of all actions in `System`. Then for any `d`$\in \mathbb{R}^+$, if the composition | `<Tester(d), System>` can evolve to a state from which it cannot perform `zzz`, then the system contains a timelock at time `d`. Notice that we are not saying that *all* executions of `System` will timelock, but rather that `System` can timelock.

This illustration indicates why timelocks represent such degenerate situations - even though the `Tester` is in all respects independent of the system, e.g. it could be that `Tester` is executed on the Moon and `System` is executed on Earth without any co-operation, the fact that the system cannot pass time prevents the tester from passing time as well. Thus, *time really does stop* and it stops everywhere because of a degenerate piece of *local* behaviour.

We can also give a semantic definition of the notion, which is similar to definitions given in [9]. However, we first need a little notation. A trace of a timed automaton A has the form, $\rho = s_0\, y_1\, s_1 \cdots s_{n-1}\, y_n\, s_n$, where, $\forall i(0 \leq i \leq n) \,.\, s_i \in [\![A]\!].1$ (throughout the paper we use the notation $t.i$ to access the ith element of a tuple); $s_0 = [l_0, v_0]$; $y_i \in \mathbb{A} \cup \mathbb{R}^+$; and

$\forall i(0 \leq i \leq n-1).\ s_i \xrightarrow{y_i+1} s_{i+1}$. We let $Tr(A)$ denote the set of all traces of A and we define the function *delay* as,

$$delay(\rho) = \Sigma\{\, y_i \mid 1 \leq i \leq n \ \wedge\ y_i \in \mathbb{R}^+ \}$$

Now we say that A can timelock at time d iff

$$\exists \rho \in Tr(A).\,(delay(\rho) < d \wedge \forall \sigma \in Tr(A).\,(\rho\, pref\, \sigma \implies delay(\sigma) < d))$$

where $\rho_1\, pref\, \rho_2$ if and only if ρ_1 is a prefix of ρ_2. Intuitively this expresses that there is a state reachable before d time units has passed, from which it is not possible for time to elapse beyond d. Notice this definition does not preclude the system evolving "while timelocked".

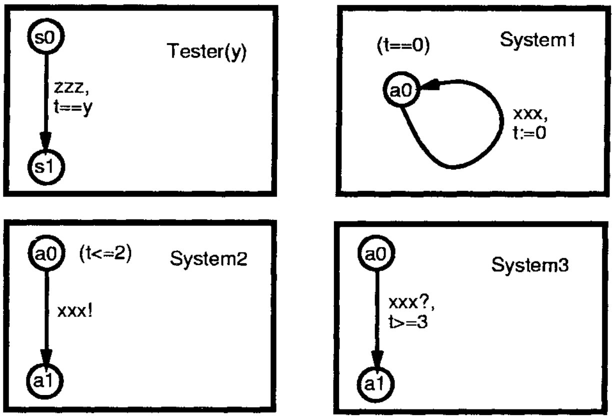

Figure 1. A Tester and Timelock Illustrations

There are two different forms of timelock:-

1 *Zeno Timelocks.* These arise when the system has an infinite behaviour but time cannot pass beyond a certain point. In other terms, an infinite number of discrete transitions are performed in a finite period of time. An example of such a specification is `System1` (see figure 1).

2 *Time Action Locks.* These are situations in which a state is reached from which neither time or action transitions can be performed. An example of such a lock is a trivial TA with no transitions and one location with invariant *false.* However, more problematically, time action locks can arise through mismatched synchronisations, e.g. | `<System2, System3>` (from figure 1) contains a timelock at

time `2`, which arises because `System2` must have performed (and thus, synchronised on) action `xxx` by the time `t` reaches `2` while `System3` does not start offering `xxx` until after `t` has past `2`.

The interesting difference between these two varieties of timelock is that the first one locks time, but it is not action locked. However, the second reaches a state in which neither time passing or action transitions are possible.

A relevant property which appears in the literature is that of time reactivity. It ensures that a system is time action lock free.

Definition 1 *A system is said to be time reactive if it can never reach a state in which neither time or action transitions can be performed.*

Action Locks. Timelocks are much more serious faults than action locks, since the latter generate *local deadlocks,* however, cannot prevent an independent process from evolving. A TA is action locked when it reaches a state from which, however long time is allowed to pass, an action will never be possible. The natural interpretation of action lock in the setting of timed systems is as follows.

Definition 2 *A state* $[l, v]$ *of a TA* A *is an action lock, denoted* $AL([l, v])$, *if and only if,* $\forall t \in \mathrm{R}^{+0}$ $([l,v + t] \in [\![A]\!].1 \;\Rightarrow\; [l,v + t] \overset{e}{\not\Rightarrow})$, *where* $[l, v + t] \in [\![A]\!].1$ *implies* $[l, v + t]$ *is reachable from* $[l, v]$ *by the definition of* $[\![\,]\!]$. *A TA* A *contains an action lock iff* $\exists s \in [\![A]\!].1 \,.\, AL(s)$.

3. TIMELOCKS

Zeno Timelocks. Using an approach of Tripakis [9] we introduce a static construction which ensures zeno timelock freeness. The idea is to ensure that for each loop in an automaton, time must pass by at least $\epsilon \in \mathrm{R}^{+}$ on every iteration (this is similar to imposing time guardedness in timed process algebra). First a definition.

Definition 3 *For* $A \in TA$ *we define a structural loop to be a sequence of distinct transitions,* $l_0 \xrightarrow{e_1,g_1,r_1} l_1 \xrightarrow{e_2,g_2,r_2} .... \xrightarrow{e_n,g_n,r_n} l_n$, *s. t.* $l_0 = l_n$.

A is called strongly non-zeno if, for every such structural loop, there exists a clock $c \in A.5$, $\epsilon \in \mathrm{R}^{+}$ *and* $0 \le i, j \le n$ *s.t.,* (*1*) $c \in r_i$; *and (2) c is bounded from below in step j, i.e.* $(c < \epsilon) \cap g_j = false$.

Clearly, `System1` of figure 1 fails to be strongly non-zeno since a suitable $\epsilon \in \mathrm{R}^{+}$ does not exist. The following result was presented in [9].

Proposition 1 *If* $A \in TA$ *is strongly non-zeno then* $Tr(A)$ *does not contain a path that is both infinite and yields a timelock.*

In addition, strong non-zenoness is well behaved through parallel composition. Specifically, the following result was presented in [9]. It ensures that we cannot generate new zeno timelocks through parallel composition.

Proposition 2 *If $A_1,...,A_n \in TA$ are strongly non-zeno then $| \langle A_1, ..., A_n \rangle$ is also strongly non-zeno.*

Also although we have no empirical evidence, in accordance with [9], we believe that in practice, specifications will invariably be strongly non-zeno.

The Nature of Synchronisation. Perhaps the most counter-intuitive aspect of the timelock story is the manner in which timelocks can arise from mis-matched synchronisations, such as the composition in figure 1. If we consider how this problem arises we can see that it is caused by the particular interpretation of urgent interaction employed in TA.

It is without doubt true that facilities to express urgency are required. However, it is our perspective that while urgency is needed, currently it is given an excessively strong formulation. We illustrate the issue with the following example.

Example. Consider the specification of the Dying Dining Philosophers problem. The scenario is basically the same as the Dining Philosophers except here we have extra constraints which state that philosophers *die* if they do not eat within certain time periods.

For example, if at a particular state, Aristotle must eat within 10 time units to avoid death, in TA his situation could be represented as state `l0` of timed automaton `Aris1` in figure 2. In addition, if say the fork he requires is being used by another philosopher, the relevant global behaviour of the rest of the system might correspond to `Rest1` in state `m0` (see figure 2 again).

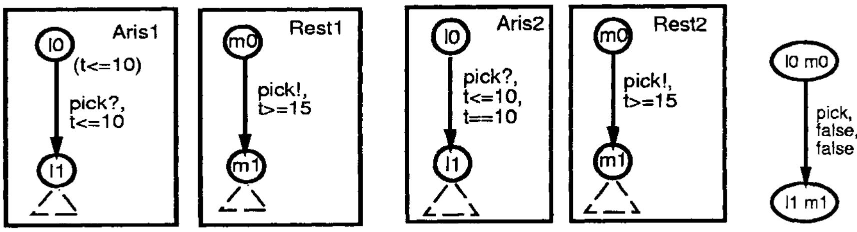

Figure 2. Dying Dining Philosophers Automata

However, the formulation | `<Aris1,Rest1>` will timelock when `t` reaches `10`. This seems counter-intuitive. Aristotle knows he must pick-up his fork by a certain time otherwise drastic consequences will result for him (this is why he "registers" his `pick` request as urgent). However, if he *locally* fails to have his requirement satisfied, he cannot *globally* prevent the rest of the world from progressing, rather a *local* deadlock should result. As a consequence Aristotle might be dead, but as we all know, "the world will go on!"

Conceptually what is happening is that Aristotle is enforcing that his `pick` action must be taken *even if it is not possible,* i.e. it is not enabled. However, we would argue that urgency can only be forced if an action is possible. In other words, it should only be possible to make an action urgent if it is enabled, i.e.

must requires may or, in other terms, you can only force what is possible.

One way in which such an interpretation of urgency has previously been obtained is through only allowing urgency to be applied to internal actions. This is the so called *as soon as possible* (asap) principle [8], much discussed in the timed process algebra community. This property indeed prevents the occurrence of timelocks due to synchronisation mismatches, but unfortunately, it is not a suitable solution for timed automata. This is because TA do not have a hiding operator. In timed process algebra with asap the hiding operator, which turns observable into internal actions, has an important role since (implicitly) it makes actions urgent. The absence of hiding in TA means that we cannot (selectively) take an observable action that results from synchronising half actions and turn it into an (urgent) internal action.

Thus, now we consider an alternative framework for TA specification - *Timed Automata with Deadlines* (TADs) which was initially devised by Bornot and Sifakis [2, 3] and with which we can obtain the synchronisation interpretation we desire. Our presentation follows that in [4], with some refinements.

TADs Basics. For a full introduction to TADs, we refer the interested reader to [2, 3]; here we highlight the main principles.

Deadlines on Transitions. Rather than placing invariants on states, deadlines are associated with transitions. In order to do this, transitions are annotated with 4-tuples, (e, g, d, r), where e is the transition label; g is the guard; d is the deadline; and r is the reset set. Conceptually, deadlines state when transitions *must* be taken and taken immediately. Since we have deadlines on transitions there is no need for invariants on states. It is also assumed that the constraint, $d \Rightarrow g$ holds, which ensures that if a transition is forced to happen it is also able to happen. As a result of this constraint, *TADs are time reactive.*

(Timewise) Priorities. By restricting guards and deadlines in choice contexts, prioritised choice can be expressed, e.g. if we have two transitions, `b1` = (`e1,g1,d1,r1`) and `b2` = (`e2,g2,d2,r2`), and we are at a state with a choice between them, then we can give `b2` priority over `b1` by restricting the guards and deadlines of `b1` to (`e1,g1',d1',r1`). The form of priority we use is to enforce the following restricted guard and deadline, `g1'` = `g1` $\wedge$ $\Box\neg$`g2` and `d1'` = `d1` $\wedge$ `g1'`, where $\Box$ is the temporal operator henceforth. This ensures that `b1` is only enabled if `g1` holds and there is *no point in the future* at which `g2` will hold.

Parallel Composition with Escape Transitions. The TADs framework employs a different parallel composition operator to that arising in standard timed automata. The key idea is that of an *escape transition.* These are the local transitions of automaton components that are combined when generating a synchronisation transition. Thus, not only are synchronisations included, but component transitions of the synchronisation are as well. The timewise priority mechanism is then used to give the synchronisation transition highest priority. Intuitively, the escape transitions can only happen if the synchronisation transition will never be enabled.

In fact, in addition to ensuring time reactivity, the TADs framework limits the occurrence of action locks. Specifically, the escape transitions allow the components of a parallel composition to escape a potential action lock by evolving locally.

We briefly review the definition of TADs. An arbitrary *TAD,* has the form, $(L, l_0, \rightarrow, C)$, where L is a finite set of locations; l_0 is the *start location;* and C is the set of clocks.

$\rightarrow \subseteq L \times \mathrm{A} \times CC_C \times CC_C \times \mathrm{P}(C) \times L$ is a transition relation. A typical element of which is, (l_1, e, g, d, r, l_2), where $l_1, l_2 \in L$; e labels the transition; g is a guard; d is a deadline; and r is a reset set. $(l_1, e, g, d, r, l_2) \in \rightarrow$ is typically written, $l_1 \xrightarrow{e,g,d,r} l_2$.

As was the case with TA, TADs are semantically interpreted as transition systems. The following two inference rules are used for this,

$$(S1)\ \frac{l \xrightarrow{e,g,d,r} l' \quad g(v)}{[l,v] \overset{e}{\Longrightarrow} [l', r(v)]} \qquad (S2)\ \frac{\forall l' \,.\, l \xrightarrow{e,g,d,r} l' \implies \forall t' < t \,.\, \neg d(v+t')}{[l,v] \overset{t}{\Longrightarrow} [l, v+t]}$$

Now we define the semantic map $[\![\,]\!]$ from TADs to transition systems as follows, $[\![(L, l_0, \rightarrow, C)]\!] = (S, s_0, \Rightarrow)$ where, $S = \{ s' \in L \times \mathrm{V}_C \mid \exists s \in S, y \in Lab \,.\, s \overset{y}{\Longrightarrow} s' \} \cup \{[l_0, v_0]\}$; $s_0 = [l_0, v_0]$; and $\Rightarrow \subseteq (L \times \mathrm{V}_C) \times Lab \times (L \times \mathrm{V}_C)$ satisfies $(S1)$ and $(S2)$. Notice that, once again, S only contains reachable states.

In addition, we will use the function:

$$\theta_B(l) = \{ (e,g,d,r) \mid \exists l' . l \xrightarrow{e,g,d,r} l' \ \wedge \ e \in B \}$$

In [4] we considered three different TADs parallel composition rules - standard TADs, Sparse TADs and TADs with Minimal Priority Escape Transitions. The first of these coincides with Bornot and Sifakis's definition [2, 3], while the latter two were developed by us. We argued in favour of the latter two, since standard TADs composition generated too many escape transitions. Here we re-iterate the Sparse TADs and TADs with Minimal Priority Escape Transitions definitions.

Sparse TADs. This is a minimal TADs approach, in which we do not generate *any* escape transitions. The following parallel composition (denoted $||^s$) rules are used:

$$\frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{||^s u \xrightarrow{x,g',d',r_i \cup r_j} ||^s u[i'/i, j'/j]} \qquad \frac{u[i] \xrightarrow{x,g,d,r} u[i]' \quad x \in CA}{||^s u \xrightarrow{x,g,d,r} ||^s u[i'/i]}$$

where $1 \leq i \neq j \leq |u|$, $g' = g_i \wedge g_j$ *and* $d' = g' \wedge (d_i \vee d_j)$.

These rules prevent uncompleted actions from arising in the composite behaviour; they only arise in the generation of completed actions, while (already) completed actions offered by components of the parallel composition can be performed independently. This definition has the same spirit as the normal TA rules of parallel composition. The difference being that here we have deadlines which we constrain during composition to preserve the property $d \Rightarrow g$, and hence to ensure time-reactivity.

Furthermore as a consequence of these characteristics of sparse TADs we have revised the interpretation of synchronisation in the manner we proposed. For example, if we consider again the Dying Dining Philosophers illustration, the obvious TADs formulation of `Aris1` and `Rest1`, are `Aris2` and `Rest2` shown in figure 2. Now sparse TADs composition of the two TADs yields the behaviour shown on the right of figure 2, which is action locked. This is the outcome that we were seeking. Since the `pick` synchronisation is not enabled, urgency cannot be enforced. This is reflected in both the guard and deadline in figure 2 being *false.*

TADs with Minimal Priority Escape Transitions. The idea here is the same as standard TADs, but rather than just giving escape transitions lower priority than their corresponding synchronisation, we also give them lower priority than other completed transitions. Thus, a component can only perform an escape transition if the component will never be able to perform a completed transition. This seems appropriate as our view of escape transitions is that they should only be performed as a very last resort - when the choice is between performing them or reaching an “error” state.

Letting, $1 \leq j \neq i \leq |u|$, the parallel composition (denoted $||^m$) rules are:

$$(R1) \quad \frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{||^m u \xrightarrow{x,g',d',r_i \cup r_j} ||^m u[i'/i, j'/j]}$$

$$(R2) \quad \frac{u[i] \xrightarrow{x,g,d,r} u[i]' \quad x \in CA}{||^m u \xrightarrow{x,g,d,r} ||^m u[i'/i]} \qquad (R3) \quad \frac{u[i] \xrightarrow{a,g,d,r} u[i]' \quad a \in HA}{||^m u \xrightarrow{a,g'',d'',r} ||^m u[i'/i]}$$

where, $g' = g_i \wedge g_j$, $d' = g' \wedge (d_i \vee d_j)$ and $1 \leq k \neq i \leq |u|$ in,

$$\begin{aligned} g'' &= g \wedge \bigwedge\{ \Box\neg q.2 \,|\, q \in \theta_{CA}(u[i]) \} \wedge \\ &\quad \bigwedge\{ \Box\neg(q.2 \wedge q'.2) \,|\, q \in \theta_{HA}(u[i]) \wedge q' \in \theta_{\{\overline{q.1}\}}(u[k]) \} \\ d'' &= d \wedge g'' \end{aligned}$$

(R1) is the normal synchronisation rule; *(R2)* defines interleaving of completed transitions; and *(R3)* defines interleaving of incomplete, i.e. escape, transitions. In this final rule, g'' holds when, (1) g holds; (2) it is not the case that an already completed transition from $u[i]$ could eventually become enabled; and (3) it is not the case that an incomplete transition (including a itself) offered at state $u[i]$ could eventually be completed. Furthermore, the definition ensures that $d \Rightarrow g$ and thus that time reactivity is preserved. In addition, we again obtain the “weaker” handling of urgency in synchronisation that we seek.

4. ACTION LOCKS

The last section and the TADs framework in general provide a means to compose automata together without generating timelocks. This then raises the issue of whether the same can be done for action locks.

It is clear that independent parallel composition (both in an untimed and a timed setting) preserves action lock freedom (see [5] for a formal justification). However, interaction free parallel composition is of limited value. Thus, here we consider how the same action lock compositionality property can be obtained but while allowing interaction between processes. Our definition builds upon TADs with Minimum Priority Escape Transitions.

Consider the parallel composition $||^a A$ where A is a vector of TADs in which the component automata have disjoint clock sets. This is necessary to avoid action locks arising as a result of component automata resetting the clocks used by other components (see [5] for further justification). Letting, $1 \leq i \neq j \leq |u|$, the product rules for $||^a A$ are,

$$(RCA)\quad \frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{||^a u \xrightarrow{x,g',d',r_i \cup r_j} ||^a u[i'/i, j'/j]}$$

$$(RIA)\quad \frac{u[i] \xrightarrow{x,g,d,r} u[i]' \quad x \in CA}{||^a u \xrightarrow{x,g,d,r} ||^a u[i'/i]} \qquad (RHA)\quad \frac{u[i] \xrightarrow{a,g,d,r} u'[i] \quad a \in HA}{||^a u \xrightarrow{a,g'',d'',r} ||^a u[i'/i]}$$

where, $g' = g_i \wedge g_j$, $d' = g' \wedge (d_i \vee d_j)$ and $1 \leq k \neq i \leq |u|$ in,

$$\begin{aligned} g'' &= (g \wedge \bigwedge\{\Box\neg q.2 \,|\, q \in \theta_{CA}(u[i])\} \wedge \\ &\quad \bigwedge\{\Box\neg(q.2 \wedge q'.2) \,|\, q \in \theta_{HA}(u[i]) \wedge q' \in \theta_{\{\overline{q.1}\}}(u[k])\}) \vee d'' \\ d'' &= d \wedge \bigwedge\{\neg q.3 \,|\, q \in \theta_{CA}(u[i])\} \wedge \\ &\quad \bigwedge\{\neg(q.2 \wedge q'.2 \wedge (q.3 \vee q'.3)) \,|\, q \in \theta_{HA}(u[i]) \wedge q' \in \theta_{\{\overline{q.1}\}}(u[k])\} \end{aligned}$$

(RCA) is the (now) familiar “conjunctive” synchronisation rule, with the deadline constraint ensuring that $d \Rightarrow g$ and thus preserving time reactivity. **(RIA)** gives the also familiar interleaved modelling of independent parallelism. **(RHA)** generates escape transitions in order to avoid action locks, with the guard and deadline constructions controlling when the escape transitions can occur. We justify our guard and deadline definitions now.

The guard in (RHA). This is a disjunction between the guard construction for escape transitions presented in section 2 and the deadline (d''). We justify the guard based disjunct (i.e. the first) here. A later point justifies disjoining with the deadline.

The basic idea of this first disjunct, is to enable the product to escape action locks resulting from mismatched synchronisations. As a simple illustration of this consider `A0` and `A1` in figure 3. Both of these TADs are action lock free. However, if just rules *(RCA)* and *(RIA)* are used the composition of `A0` and `A1` will action lock immediately as neither synchronisation can be fulfilled. However, application of rule *(RHA)* in conjunction with *(RCA)* and *(RIA)* will allow the action lock to be escaped, as shown in composition (i) in figure 3.

The deadline in (RHA). The definition of d'' has a similar shape to the guard construction we just considered, however, the temporal operators are not included. The construction states that, the deadline (d'') of the escape transition holds if and only if,

1 the deadline of the corresponding component transition (d) holds;

2 no internal transition of the component is at its deadline; and

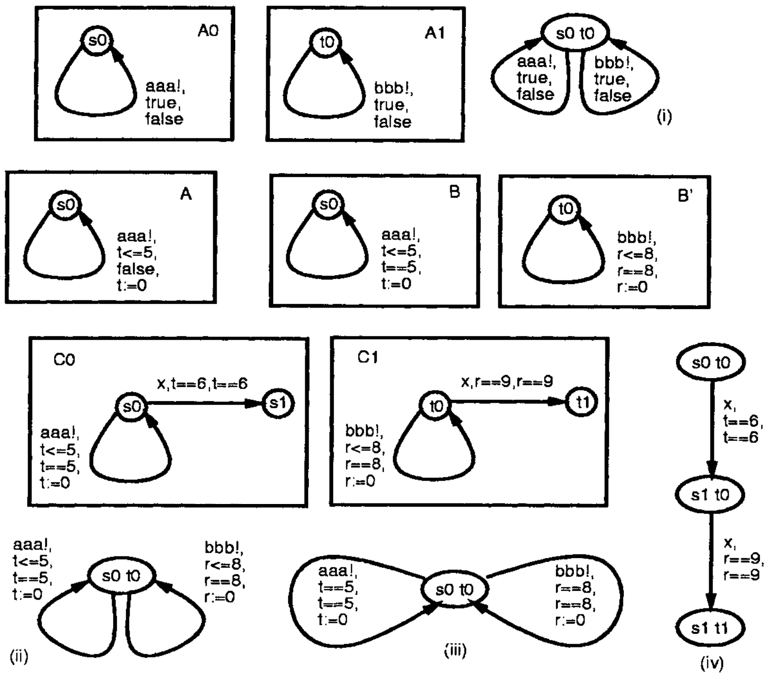

Figure 3. Assorted TADs

3 no synchronisation which includes a half action of the component is at its deadline.

The intuition behind the rule is that any (non competing) deadline that appears in the component but that does not arise in the product (because of a failed synchronisation) has its deadline preserved in an escape transition of the product. A deadline of a transition is *competing* at a state if the deadline of an alternative transition also holds there.

This deadline construction is motivated by the observation that in the majority of cases it is the deadline that ensures action lock freeness. For example, although the automaton A in figure 3 is strongly connected it is not action lock free. In particular, assuming s0 is first entered with t==0, if it stays in state s0 for longer than 5 time units, it will action lock. Furthermore, there is nothing constraining the length of time the automaton can idle in state s0 as the deadline of the aaa! transition is false. However, (assuming s0 is entered with t<=5) if the *false* deadline is replaced by, say, t==5, then it would be action lock free.

Now in order to obtain the action lock freeness property that we desire we need to guarantee that deadlines that ensure action lock freeness of component automata are preserved in the product (either through appearing as a result of rules *(RCA)* or *(RIA)* or by including relevant escape transitions). Our rule does this. Consider the two action lock free automata B and B' shown in figure 3. With just rules *(RCA)* and *(RIA)* the product of B and B' would be action locked. However, with *(RHA)* as well, the product automaton (ii) shown in figure 3 would result.

In fact, this product would have resulted from application of the rules presented in section 2 where the deadline is simply $d'' = d \wedge g''$. However, the example in figure 3 of two more action lock free TADS (`C0` and `C1`) shows that this is not sufficient in the general case. This is because according to the rules of section 2, the parallel composition of `C0` and `C1` would be as shown in figure 3 (iv) which will action lock at state `s1 t1.`

The problem is that the guards of the aaa! and bbb! escape transitions that the rules of section 2 generate, are false. This is because in both automata an internal action can eventually be taken and this internal action will take priority.

However, if we apply the rules *(RCA), (RIA)* and *(RHA)* of $||^a$ then a product "behaviourally equivalent" to figure 3 (iii) results. This is because the deadline prevents clock t passing 5 and clock `r` passing 8. Notice that the guard has been pruned to match the deadline. This ensures that the enabling of aaa! and bbb! is minimised to only what is required to preserve the desired action lock freeness property.

Also notice that this example illustrates why the priority enforced in the deadline has to be immediate and including temporal operators is inappropriate. Specifically, if a deadline d ensures action lock freedom then even if later transitions are possible the deadline must be preserved exactly in the product in order to prevent later transitions from being enabled which allow an action lock to be reached, e.g. the internal transitions in `C0` and `C1` above. This may not be the most refined solution since we might add an escape transition even though a later transition may prevent the action lock. However, it is not currently clear how to improve upon the approach.

Finally, we need to disjoin the deadline in g'' in order to ensure that $d \Rightarrow g$ and thus to preserve time reactivity. For example, without such a disjunct, the product of `C0` and `C1` would timelock when t reaches 5 as the guard on the aaa! transition from `s0 t0` would be *false.*

The central result of this section is given in the next theorem, it states that $||^a$ preserves action lock freeness. The proof of this result is not straightforward and due to space considerations it is not possible to include it here. However, the necessary theory, the full proof and illustration of use of the operator can be found in [5], which is available on the WWW.

Theorem 1
$\exists i (1 \leq i \leq |A|). A[i]$ *is action lock free* $\Rightarrow ||^a A$ *is action lock free.*

5. CONCLUSIONS

This paper builds from [4] by refining its timelock results and extending the results to action lock freeness. Although related to the work of Bornot and Sifakis [2, 3], our work is different. In particular, giving escape transtions lower priority than completed actions of a particular component, is unique to our work. Furthermore, such an interpretation is important since as we have argued, real-time structures such as timeouts are inappropriately expressed with the standard TADs parallel composition operator. In addition, in obtaining our compositionality results, the only constraint we impose on component automata is that they are time and / or action lock free. In contrast, Bornot and Sifakis require a number of well behavedness criteria to hold. This limits the generality of their approach when compared with ours.

Acknowledgements. The author has benefited greatly from discussions with Sebastian Bornot, Joseph Sifakis and Stavros Tripakis.

REFERENCES

[1] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, and Paul Pettersson amd Wang Yi. Uppaal - a tool suite for automatic verification of real-time system. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems,* 1995.

[2] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *Hybrid Systems: Computation and Control,* LNCS 1386, pages 49–63, 1998.

[3] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality, COMPOS'97,* LNCS 1536, 1997.

[4] H. Bowman. Modelling timeouts without timelocks. In *ARTS'99, Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop,* LNCS 1601, pages 335–353. Springer-Verlag, 1999.

[5] H. Bowman. On time and action lock free description of timed systems. Technical Report 16-99, Computing Laboratory, University of Kent at Canterbury, 1999. available at http://www.cs.ukc.ac.uk/people/staff/hb5/pubs.local.

[6] H. Bowman, G. Faconti, J-P. Katoen, D. Latella, and M. Massink. Automatic verification of a lip synchronisation algorithm using UPPAAL. *Formal Aspects of Computing,* 10(5-6):550–575, August 1998.

[7] C.Daws, A.Olivero, S.Tripakis, and S.Yovine. The tool KRONOS. In *Hybrid Systems III, Verification and Control,* LNCS 1066. Springer-Verlag, 1996.

[8] T. Regan. Multimedia in temporal LOTOS: A lip synchronisation algorithm. In *PSTV XIII, 13th Protocol Spec., Testing & Verification.* North-Holland, 1993.

[9] S. Tripakis. Verifying progress in timed systems. In *ARTS'99, Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop,* LNCS 1601. Springer-Verlag, 1999.